# Introduction to Python

Master 4.0

# HISTORY

Python was created by **Guido van Rossum,** and first released on **February 20, 1991** as Python 0.9.0.

While you may know the python as a large snake, the name of the Python programming language comes from an old BBC television comedy sketch series called **Monty Python's Flying Circus**.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language.

**Python 2.0** was released in **2000. Python 3.0** released in **2008**, was a major revision not completely backward-compatible with earlier versions. Python latest version 3.11.2 released on Feb 8, 2023.

**Python consistently ranks as one of the most popular programming languages.**

# PYTHON



Python is a **general purpose, open source, high-level programming language** and also provides **number of libraries** and frameworks. Python has gained popularity because of its **simplicity, easy syntax and user-friendly** environment.

Python is **strongly typed** (i.e. types are enforced), dynamically, **implicitly typed** (i.e. you don't have to declare variables), **case sensitive** (i.e. ybi and YBI are two different variables) and **object-oriented** (i.e. everything is an object).

# REASONS THAT MAKE PYTHON GREAT FOR LEARNING

- **It is easy to learn** – the time needed to learn Python is shorter than for many other languages; this means that it's possible to start the actual programming faster
- **It is easy to use for writing new software** – it's often possible to write code faster when using Python
- **It's open source**, is easy to obtain, install and deploy – Python is free, open and multiplatform; not all languages can boast that.
- Some languages require you to modify code to run on different platforms, but Python is a **cross-platform language**, which means you can run the same code on any operating system with a Python interpreter.
- It's extendable. Python code can be written in other languages (such as C++), and users can add low-level modules to the Python interpreter to customize and optimize their tools.
- It has a **multiple standard library**. This library is available for anyone to access and users don't have to write code for every single function—they can access built-in modules that help with issues in everyday programming and more.

# USAGE OF PYTHON

- Desktop Applications
- Web Applications
- Data Science
- Artificial Intelligence
- Machine Learning
- Scientific Computing
- Robotics
- Internet of Things (IoT)
- Gaming
- Mobile Apps
- Data Analysis and Pre-processing

# COMPANIES USING PYTHON

- Google (Components of Google spider and Search Engine)
- Yahoo (Maps)
- YouTube
- Mozilla
- Dropbox
- Microsoft
- Cisco
- Spotify

# Introduction Google Colab

https://colab.research.google.com/

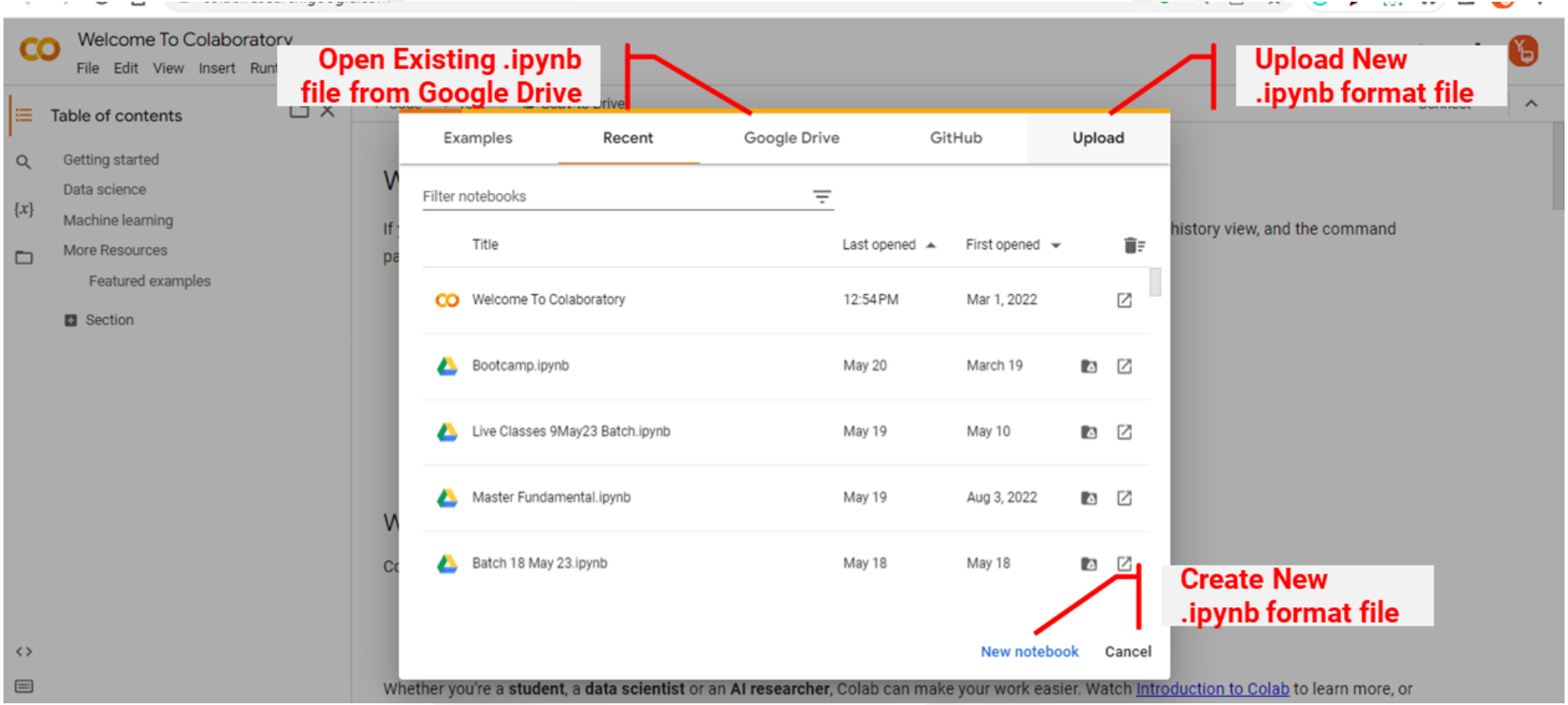
# GOOGLE COLAB

- Google Colab is an online representation of Jupyter Notebook.
- While Jupyter Notebook needs installation on a computer and can uses local machine resources, Colab is a full-fledged cloud solution for Python coding.
- Google Colab allows anybody to write and execute python code through the browser on laptop or mobile.
- You can write Python codes using Colab on your Google Chrome or Mozilla Firefox or any other web browser.

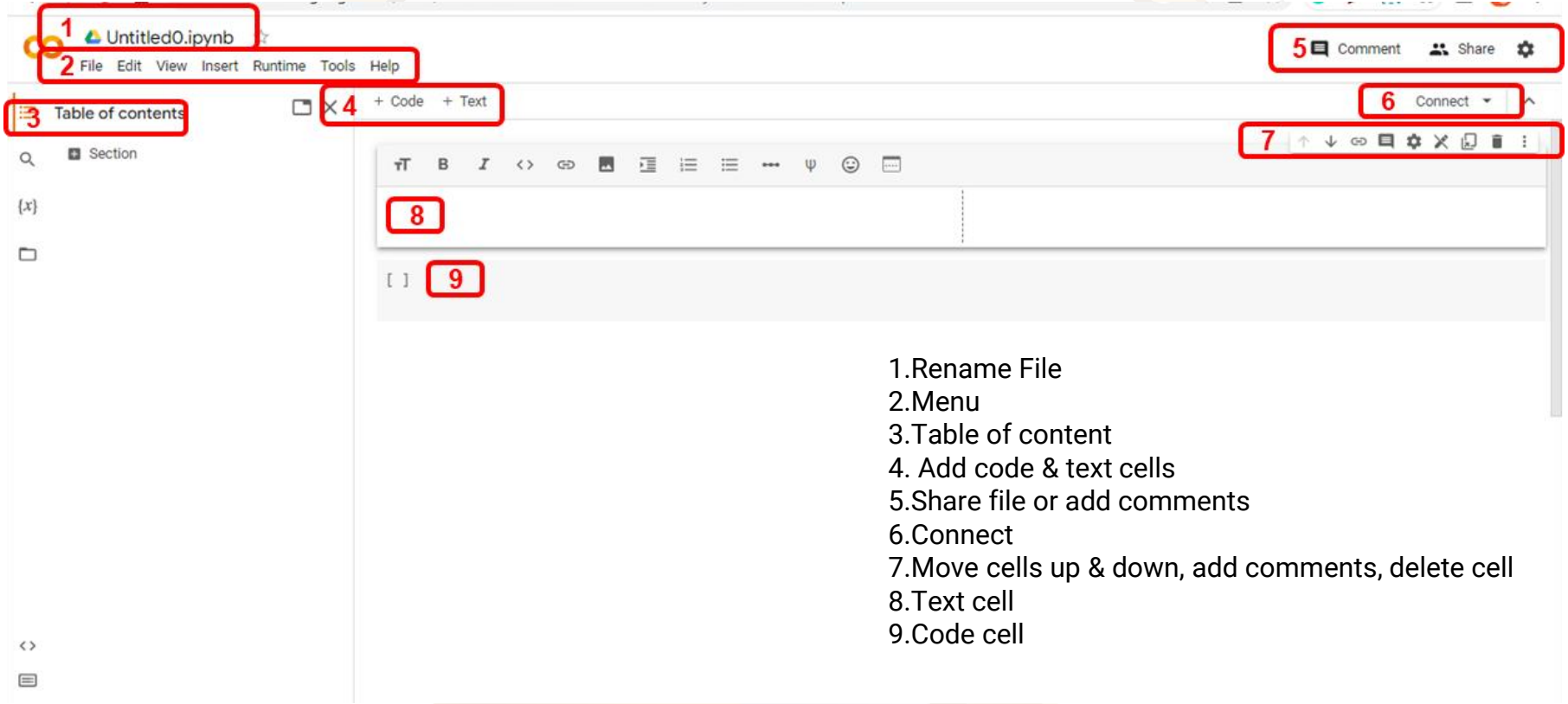
# HOW TO OPEN GOOGLE COLAB

- **Step 1:** Open any browser
- **Step 2:** Login to your Google account
- **Step 3:** Search for Colab in google search
- **Step 4:** Click or open URL ' https://colab.research.google.com/ '
- **Step 5:** Click on New Notebook option in bottom right of pop up or from File menu

# How to Create New Notebook



# GOOGLE COLAB LAYOUT




1.Rename File
2.Menu
3.Table of content
4. Add code & text cells
5.Share file or add comments
6.Connect
7.Move cells up & down, add comments, delete cell
8.Text cell
9.Code cell

# Python Variables

*A Python variable is a symbolic name that is a reference or pointer to an object. Once an object is assigned to a variable, you can refer to the object by that name. But the data itself is still contained within the object.*

# WHAT IS A VARIABLE

A variable is any characteristic, number, or quantity that can be measured or counted.

- Age (17, 25, 42, ...)
- Gender (male, female)
- Income (₹ 20000, ₹ 35000, ₹ 45000, ...)
- House price (₹ 350000, ₹ 570000, ...)
- Country of birth (India, Russia, USA, ...)
- Eye colour (brown, green, blue, ...)
- Vehicle make (TATA, Maruti, ...)

# VARIABLES

- A variable is used to store information that can be referenced later.

- We can assign data and outputs to a variable and can create an unlimited number of variables, but we have to make sure to give each variable a unique name.

# DEFINE A VARIABLE

Variables are used to store information and represent your data input.

```
In [1]: x = 5

In [2]: x
Out[2]: 5
```

We defined a **variable *x*** that is equal to the value of 5 and then ask the computer to tell the value of that variable.
Type *x* equals 5, i.e. write *x* and then execute the cell, and we get the output: 5.

```
In [3]: y = 8

In [5]: print (y)
        8
```

We assigned to y a value 8 and use the *print()* function as print(y), the machine will simply execute this command and provide the output as value of *y*.

# STRINGS

```
[1]  'Ybi Foundation'
```

```
'Ybi Foundation'
```

```
[2]  "Ybi Foundation"
```

```
'Ybi Foundation'
```

```
[3]  print('Ybi Foundation')
```

```
Ybi Foundation
```

```
[4]  ybi = 'Ybi Foundation'
```

```
[5]  ybi
```

```
'Ybi Foundation'
```

```
[6]  print(ybi)
```

```
Ybi Foundation
```

Type single or double quotation marks around the name Ybi Foundation. Python displays 'Ybi Foundation' if you don't use the print() function. Should you use print(), the output will be shown with no quotes – you'll be able to see plain text.

# VARIABLE NAMING RULES

Python allows Names to be a combination of

- lowercase letters (a to z) or
- uppercase letters (A to Z) or
- digits (0 to 9) or
- an underscore (_).

Names are case sensitive and cannot start with a number.

Names in Python cannot use any special symbols like !, @, #, $, % etc.

They can contain letters, numbers, and underscores.

ybi  YBi  _ybi  _2_ybi_  ybi_2  YbI

# Allowed Variable Names

```
[ ]  variable_name = 'abc'
```

```
[ ]  _variablename_ = 2
```

```
[ ]  variable1 = True
```

# Not Allowed Variable Names

```
[ ]  1variable = 2
```

```
  File "<ipython-input-11-672ad970f03b>", line 1
    1variable = 2
             ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

```
[ ]  @variable = 2
```

```
  File "<ipython-input-12-7294ab471b8a>", line 1
    @variable = 2
              ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

# Reserved Key Words for Variable Names

- Reserved words (also called keywords) are defined with predefined meaning and syntax in the language.
- These keywords have to be used to develop programming instructions.
- Reserved words can't be used as identifiers for other programming elements like name of variable, function etc.

# Reserved Key Words for Variable Names

```
[ ]  class = 2

  File "<ipython-input-13-ed55d338b5c6>", line 1
    class = 2
          ^
SyntaxError: invalid syntax

SEARCH STACK OVERFLOW
```

['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', '**class**', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

# SYNTAX

- Python has no mandatory statement termination characters and blocks are specified by indentation.
- Indent to begin a block, dedent to end one.
- Statements that expect an indentation level end in a colon (:).

# ADD COMMENTS

```
[1]  # This is comment line not run as code

     1+1

2
```

```
[2]  # comment 1
     # comment 2

     1+1

2
```

- Comments are sentences not executed by the programming language i.e. not read them as instructions.
- Put a hash sign (#) at the beginning of each line to work as a comment.
- For comments on two or more lines, place the hash sign at the beginning of each line.
- Comments start with the pound (#) sign and are single-line, multi-line strings are used for multi-line comments.

# LINE CONTINUATION

```
[1]  2.0 + 3.0\
     - 1.0
```

```
4.0
```

```
[2]  2.0 + 3.0\
     - 1.0 \
     - 2
```

```
2.0
```

- To write code in multiple lines, so that program read it as one command.
- This could be achieved by putting a back slash (\) at the end of the first line. It indicates the continuation of the same code on a new line.

# INDEXING

Python is a zero index language

```
[1]  ybif = 'Ybi Foundation'

[2]  ybif
'Ybi Foundation'

[3]  ybif[0]
'Y'

[4]  ybif[1]
'b'

[5]  ybif[-1]
'n'
```

Use square brackets and specify the position/index of the letter to extract.

| Y | b | i | | F | o | u | n | d | a | t | i | o | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

# STRUCTURE YOUR CODE WITH INDENTATION

```
def sqrt(x):
    sqrt = x*x
    return(sqrt)
```

```
[2] sqrt(2)
```

```
4
```

- Indentation is important as it will define the hierarchy of codes for execution in python.
- In the example code, def and return clearly in distinguishable blocks of code or blocks of commands.
- Everything that belongs to same function is written with one indentation to the inside. Once you decide to code something new, start on a new line with no indentation.
- The indented blocks of code are clear and visibly clarifies the logic to solve problem.

# THE DOUBLE EQUALITY SIGN

```
[1]  x = 2*3
     x
```

```
6
```

```
[2]  y = 2**3
     y
```

```
8
```

```
[3]  x == 6
```

```
True
```

```
[4]  y == 6
```

```
False
```

```
[5]  y == 8
```

```
True
```

- The equals sign means “assign” or “bind to”.
- For instance, assign 2 to the power of 3 to a variable y. Hence y, will be equal to 8.
- To check equality between values, use double equality sign. With possible outcomes as True or False.
- Values are assigned (in fact, objects are bound to names) with the equals sign (“=”), and equality testing is done using two equals signs (“==”).

# REASSIGN VALUES

```
[1]  x = 1
     x
```

```
1
```

```
[2]  x = 3
```

```
[3]  x +2
```

```
5
```

```
[4]  x
```

```
3
```

- If you assign the value of 1 to a variable x, your output after executing x will be 1.
- After that, if you assign 3 to the same variable x, x will be equal to 3, not 1 anymore.
- Python reassigns values to its objects. Therefore, remember the last command is valid, and older commands are overwritten.

# HELP

```
[1]  help(len)
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

```
[2]  len.__doc__
```

```
'Return the number of items in a container.'
```

```
[3]  dir()
```

```
['In',
 'Out',
 '_',
 '_2',
 '__',
 '___',
 '__builtin__',
 '__builtins__',
 '__doc__',
```

- Help in Python is available right in the interpreter.
- To know how an object works use help(<object>)
- Also useful are dir(), which shows you all the object's methods
- <object>.__doc__ shows documentation string

# Python Data Types

# PYTHON LANGUAGE DATA TYPES

- Numerical Data
  1. Integer
  2. Float
  3. Complex
- Boolean
- Date time *(A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.)*
- Sequence Data
  1. str
  2. bytes
  3. bytearray
  4. Range
- None Type

# ARITHMETIC OPERATORS

| Name | Operator | Example | |
|---|---|---|---|
| Addition | + | x + y | 10+3 = 13 |
| Subtraction | - | x - y | 10-3 = 7 |
| Multiplication | * | x * y | 10*3 = 30 |
| Division (float division) | / | x / y | 10/3 = 3.33 |
| Division (floor division) | // | x // y | 10//3 = 3 |
| Modulus (return the reminder) | % | x % y | 10%3 = 1 |
| Exponentiation | ** | x ** y | 10**3 = 1000 |

# COMPARISON OPERATORS

| Name | Operator | Example |
|---|---|---|
| Equal | == | x == y |
| Not equal | != | x != y |
| Greater than | > | x > y |
| Less than | < | x < y |
| Greater than or equal to | >= | x >= y |
| Less than or equal to | <= | x <= y |

# LOGICAL OPERATOR

| Description | Operator | Example |
|---|---|---|
| Returns True if both statements are true | and | x < 15 and x < 20 |
| Returns True if one of the statements is true | or | x < 15 or x < 20 |
| Reverse the result, returns False if the result is true | not | not(x < 15 and x < 20) |